Варіант 7

Умова:

№гр: 43 **№сп:** 7 Go !

$A =$

7.03	1.16	0.91	1.135
1.16	3.39	1.3	0.16
0.91	1.3	6.21	2.1
1.135	0.16	2.1	5.33

Реалізувати метод Крилова

Мета роботи:

засвоїти ітераційні методи, які приходять на допомогу, якщо прямі методи застосувати неможливо.

Завдання:

- Розв'язати часткову проблему власних значень: знайти найбільше та найменше власні числа та відповідні їм власні вектори степеневим методом або методом скалярних добутків.
- Розвязати повну проблему методом Крилова.

Текст програми (одразу з результатами):

```
import numpy as np
from sympy.solvers import solve
from sympy import Symbol, re
l = Symbol('lambda')
import pdb

A = np.matrix([[7.03, 1.16, 0.91, 1.135],
               [1.16, 3.39, 1.30, 0.16],
               [0.91, 1.30, 6.21, 2.10],
               [1.135, 0.16, 2.10, 5.33]])
eigs, eigvs = np.linalg.eig(A)
print "Eigenvalues: \n", eigs
print
print "Eigenvectors: \n", eigvs
```

Eigenvalues:

```
[ 9.38114919  6.02074842  2.47241469  4.0856877 ]
```

Eigenvectors:

```
[[ 0.58965153  0.77843754 -0.20746804  0.05747249]
 [ 0.25602053  0.06013783  0.80667063 -0.52926309]
 [ 0.59424973 -0.53271786 -0.42750033 -0.42464386]
 [ 0.48335475 -0.32654269  0.35140145  0.7322944 ]]
```

```
y = []
#create rand vector
y.append(np.random.normal(size=A.shape[0]))
y[0] = y[0].reshape(4,1)
#and now from rand vector we created other 3 vectors (Krylov method)
for i in range(1,A.shape[0]+1):
    y.append(A.dot(y[i-1].reshape(4,1)))

#last element
m = y[-1]
#
M = np.hstack(y[0:-1:])
print "Now we have to solve the equation M*x = -m, where:"
print "1) m: \n", m, "\n"
print "2) matrix f the system M: \n", M
```

```
Now we have to solve the equation M*x = -m, where:
1) m:
[[ 7315.35569953]
 [ 2888.6538618 ]
 [ 5923.58619717]
 [ 4762.13427945]]

2) matrix f the system M:
[[  1.74332033e+00   1.24181917e+01   9.88285042e+01   8.35191476e+02]
 [ -3.30372095e-01   2.27318355e+00   3.17604588e+01   3.10893580e+02]
 [  1.10431080e+00   7.16538018e+00   6.31399971e+01   6.08668066e+02]
 [ -4.04441754e-01   2.08918718e+00   4.06410230e+01   4.66462672e+02]]
```

```
coefs = np.linalg.solve(M,-m)
p1,p2,p3,p4 = np.squeeze(np.asarray(coefs))
print "Actually, we solved the equation and now have the results, which by coincidence :) are our parameters"
print "of characteristic polynom: "
print "p1 = ", p1
print "p2 = ", p2
print "p3 = ", p3
print "p4 = ", p4
p = [p1,p2,p3,p4]
```

```
Actually, we solved the equation and now have the results, which by coincidence :)
are our parameters
of characteristic polynom:
p1 =  570.549075532
p2 =  -525.994206
p3 =  167.590275
p4 =  -21.96
```

```
print "Yeeeah, eventually solving that equation! :)"
eigvals = map(re, solve(l**4 + p4*l**3 + p3*l**2 + p2*l + p1, l))
```

Yeeeah, eventually solving that equation! :)

```
print "Eigenvalues (Krylov method): \n", eigvals
```

```
Eigenvalues (Krylov method):
[2.47241469431624, 4.08568770002531, 6.02074841751810, 9.38114918814035]
```

```
#define the function for finding eigenvectors by eigenvalues
def vec(M, lmd):
    g3 = np.random.randn()
    g2 = (lmd+p4)*g3
    g1 = lmd*g2 + p3*g3
    g0 = lmd*g1+ p2*g3
    print lmd * g0 + p1 * g3 < 1e-5
    return M[:,0]*g0 +  M[:,1]*g1 + M[:,2]*g2 + M[:,3]*g3

eigenvectors_krylov=[None for x in range(len(eigvals))]
for i in range(len(eigvals)):
    eigenvectors_krylov[i]=vec(M, float(eigvals[i]))
    eigenvectors_krylov[i]= eigenvectors_krylov[i]/np.linalg.norm( eigenvectors_krylov[i])
    print "Eigenvector (Krylov method) #", i+1,":\n", eigenvectors_krylov[i]
```

```
True
Eigenvector (Krylov method) # 1 :
[[-0.20746804]
 [ 0.80667063]
 [-0.42750033]
 [ 0.35140145]]
True
Eigenvector (Krylov method) # 2 :
[[ 0.05747249]
 [-0.52926309]
 [-0.42464386]
 [ 0.7322944 ]]
True
Eigenvector (Krylov method) # 3 :
[[ 0.77843754]
 [ 0.06013783]
 [-0.53271786]
 [-0.32654269]]
True
Eigenvector (Krylov method) # 4 :
[[-0.58965153]
 [-0.25602053]
 [-0.59424973]
 [-0.48335475]]
```

Start of using POWER METHOD
He-he, finally :)

```
#max lambda

def compute_high(Z, count):
    v_max = np.random.randn(Z.shape[0]).reshape(Z.shape[0],1)
```

```
    lmd_max = 0
    for i in xrange(200000):
        count+=1
        if (np.abs((Z - np.eye(Z.shape[0]) * lmd_max).dot(v_max)) < 1e-5).all():
            break
        chi = Z.dot(v_max)
        lmd_max = np.linalg.norm(chi)/np.linalg.norm(v_max)
        v_max  = chi / np.linalg.norm(chi)
    #defining whether lambda will be <0 or >0
    if (np.abs(Z.dot(v_max) - np.abs(lmd_max)*v_max) < 1e-5).all():
        lmd_max = np.abs(lmd_max)
    else:
        lmd_max = - np.abs(lmd_max)
    return (lmd_max, v_max, count)

lmd_max, v_max, count = compute_high(A, 0)

print 'Number of iterations: ', count
print
print "Lambda max: \n", lmd_max
print
print "Eigenvector for lambda max: \n", v_max
print
print "Residual vector: \n", A*v_max — lmd_max*v_max
```

Number of iterations: 33

```
Lambda max:
9.38114918807

Eigenvector for lambda max:
[[-0.589654  ]
 [-0.25602072]
 [-0.59424804]
 [-0.48335371]]

Residual vector:
[[  8.31585555e-06]
 [  6.42413404e-07]
 [ -5.69096770e-06]
 [ -3.48841012e-06]]
```

```
#low lambda
B = A - np.eye(A.shape[0])*lmd_max
lmd_B, v_min, count = compute_high(B, 0)
lmd_min = lmd_B + lmd_max
print 'Number of iterations: ', count
print
print "Lambda min: \n", lmd_min
print
print "Eigenvector for lambda min: \n", v_min
print
print "Residual vector: \n", A*v_min-lmd_min*v_min
```

Number of iterations: 200000

```
Lambda min:
2.47241469432

Eigenvector for lambda min:
[[ 0.20746804]
 [-0.80667063]
 [ 0.42750033]
 [-0.35140145]]

Residual vector:
[[ -1.11022302e-16]
 [  1.11022302e-15]
 [  2.22044605e-16]
 [ -1.11022302e-16]]
```

Висновки:

в моєму випадку метод Крилова працював швидше і краще за ступеневий метод, і давав результати більшої точності. І, власне, метод Крилова виявився кращим, адже дає змогу одразу знайти всі власні числа та відповідні їм вектори, а степеневим метод потрібно поокремо шукати мінімальне власне число і відповідний вектор, максимальне, і проміжні значення (пари число/вектор).

В результаті виконання роботи я засвоїв ітераційні методи, які приходять на допомогу, якщо прямі методи застосувати неможливо.